ГЕТЕРОГЕННАЯ МНОГОЯДЕРНАЯ ОПЕРАЦИОННАЯ СИСТЕМА: АРХИТЕКТУРА, ПОДХОДЫ, ПРЕИМУЩЕСТВА

Е. И. Клименков

Белорусский государственный университет информатики и радиоэлектроники
Минск, Беларусь
E-mail: Evgeny.Klimenkov@gmail.com

Бурное развитие процессоров, массовый переход на многоядерные процессоры и тенденции, указывающие на расцвет гетерогенных компьютерных систем в ближайшем будущем, делают актуальной разработку архитектуры операционной системы оптимизированной для выполнения в среде современных и перспективных многоядерных гетерогенных компьютерных систем.

Ключевые слова: операционная система, микроядро, гетерогенная компьютерная система, многоядерный процессор.

Внутренняя архитектура современного компьютера напоминает распределенную сетевую систему, состоящую из процессорных ядер, кэшей, внутренних связей, устройств ввода/вывода и карт расширения. При этом развитие компьютерных систем имеет следующие тенденции: переход на многоядерные процессорные устройства, бурное развитие встроенных компьютерных систем, увеличение степени интеграции между компонентами компьютерной системы, увеличение степени интеграции между компьютерными системами [1].

Технологии современных компьютеров и процессорных устройств, а также тенденции и перспективы их развития предполагают разработку и использование новых технологий организации вычислений. И, видимо, они будут представлять собой симбиоз технологий организации вычислений на центральных процессорных устройствах, массивно-параллельных и иных специализированных процессорах, технологий параллельных и распределенных вычислений, а также сетевых технологий. Постепенный массовый переход на многоядерные процессорные устройства является новым поводом к пересмотру архитектуры операционной системы, в которой нужно учесть как накопленный опыт в теории операционных систем, так и накопленные инновации в архитектуре процессоров и тенденции их дальнейшего развития. Архитектура гетерогенной многоядерной операционной системы родилась в ответ на изменение архитектуры компьютерных систем и разрабатывалась с учетом выявленных тенденций их дальнейшего развития.

Основной идеей положенной в архитектуру гетерогенной многоядерной операционной системы является разделение традиционного централизованного монолитного ядра на децентрализованное множество микроядер, обладающих высокой степенью автономности. При этом общий объем оперативной памяти компьютерной системы также разделяется между микроядрами.

Одним из ключевых принципов, используемых при проектировании и реализации ядра гетерогенной многоядерной операционной системы, является принцип минимализма. Этот принцип был окончательно сформулирован Йоханом Лидтке, который утверждал, что «функционал должен реализовываться в микроядре, только если его реализация вне микроядра, то есть реализация аналогичного функционала за пределами ядра, предотвращает реализацию некоторой необходимой функциональности системы» [2].

Другой принцип, используемый в архитектуре ядра, предлагает рассматривать ядро операционной системы как драйвер процессора. Согласно этому принципу:

1. Ядро рассматривает компьютерную систему только как подсистему процессор-память.

2. Ядро полностью скрывает от остального программного обеспечения подробности реализации всех частей процессора и входящих в его состав компонентов.

3. Определенное ядро операционной системы может быть оптимизировано для определенного процессора.

4. Операционная система поставляется с набором ядер.

Основные функции, выполняемые экспериментальным ядром гетерогенной многоядерной операционной системы, включают обработку исключений и прерываний, реализацию многозадачности, базовое управление памятью и аппаратным обеспечением, организацию межзадачной и межъядерной коммуникации.

Новой функцией, привносимой архитектурой многоядерной операционной системы, является межъядерная коммуникация. Эффективная межъядерная коммуникация может быть реализована только с одновременным использованием двух типов коммуникационных каналов, построенных на базе оперативной памяти с единым адресным пространством, характерной для многоядерных и многопроцессорных систем:

- коммуникационного канала для одноадресной передачи данных между двумя ядрами операционной системы;
- широковещательного коммуникационного канала для реализации общесистемных извещений.

Одноадресный коммуникационный канал должен обеспечить быструю и гибкую передачу сообщений. Этот тип канала должен использоваться в качестве основного канала связи, чтобы разгрузить трафик с широковещательного коммуникационного канала, который по определению является менее масштабируемым.

Реализация межъядерной коммуникации предполагает внесение в классическую модель коммуникации, состоящую из таких элементов, как участники коммуникации, канал и сообщение, еще одного элемента – области коммуникации. Данный элемент необходим для определения множества доступных участников и служит глобальным описателем существующих в системе коммуникационных каналов. В связи с тем что структура данных, представляющая область коммуникации, совместно используется всеми ядрами ОС в системе, она потенциально является одним из наиболее слабых мест в системе с точки зрения отказоустойчивости и может существенно ограничить динамическую гибкость операционной системы. Данное обстоятельство делает обоснованной динамическую распределенную реализацию данной структуры данных.

Реализация межъядерной коммуникации в архитектуре гетерогенной многоядерной операционной системы предполагает, что участники коммуникации на макроуровне будут представлены ядрами операционной системы и на микроуровне –

представлены задачами. Специальная роль коммуникационного сервера была включена в архитектуру гетерогенной многоядерной операционной системы. Задача, которая играет эту роль, представляет ядро непосредственно во время межъядерной коммуникации, и, соответственно, получает все сообщения, передающиеся ядру, и переправляет их задачам, исполняющимся в контексте ядра. Необходимо отметить, что роль коммуникационного сервера может быть динамически передана от одной задачи к другой.

Одно из ключевых понятий, которое было принесено в мир операционных систем операционной системой UNIX, было понятие паники ядра [3]. Разработчики UNIX, работавшие над операционной системой Multics, отмечают, что на тот момент более половины кода Multics было связано с восстановлением работоспособности системы после ошибок, возникших в ядре, поэтому они приняли решение о том, что UNIX должен только фиксировать ошибку, возникшую в ядре, уведомлять об этом пользователя и затем переходить в бесконечный холостой цикл. В соответствии с этой стратегией после этого пользователь должен произвести все необходимые ему действия и перезагрузить систему. Стратегия обработки ошибок ядра UNIX и такое понятие, как паника ядра, являются одними из самых важных концепций, созданных командой разработчиков UNIX.

Концепция паники ядра поддерживается архитектурой гетерогенной многоядерной операционной системы на микроуровне, но эта архитектура открывает возможность возвратиться к более интеллектуальной обработке и восстановлению после ошибок ядра на макроуровне. При разработке стратегии обработки ошибок ядра мы уже были знакомы с идеями, предложенными разработчиками Hewlett-Packard [4], которые предлагали использовать очень сложную аппаратно-программную систему для соединения по крайней мере двух серверов таким способом, который обеспечит возможности для обработки и восстановления после ошибок ядра. Однако архитектура гетерогенной операционной системы предполагает, что все ядра в системе работают с одной и той же физической памятью, и, более того, они искусственно ограничивают свой доступ к памяти, чтобы избежать интерференции. В результате такая архитектура ОС обеспечивает для разработчиков систем возможность реализовать системные службы восстановления после ошибок ядра непривилегированного режима, которые могут обеспечить беспрецедентную отказоустойчивость компьютерной системы как программного комплекса.

Чтобы проиллюстрировать большие возможности по обеспечению отказоустойчивости, представляемые архитектурой гетерогенной многоядерной операционной системы, далее будет описан один из сценариев восстановления после ошибки ядра. При этом сразу же стоит отметить, что предложенная архитектура позволяет создавать такие компьютерные системы, в которых полный отказ компьютерной системы может быть вызван только за счет отказа аппаратного обеспечения.

Давайте предположим, что одно из ядер операционной системы содержит некоторую ошибку, которая в какой-то момент времени приводит к краху ядра. Сразу же после того, как крах был зафиксирован ядром, оно переключается в режим паники ядра. В этом режиме ядро извещает систему о своем крахе и зацикливает само себя (чтобы сделать пример более простым и наглядным, мы опустили некоторые детали). Отметим, что при этом кадр памяти разрушенного ядра все еще содержит полный снимок данных, принадлежащих отказавшему ядру и всем задачам, который работал в его контексте. Остальная часть системы после этого получения сообщения о панике решает, какое из все еще работающих ядер предоставит услугу восстановле-

ния для разрушенного ядра. Ядро, которое в результате было выбрано в качестве сервера восстановления, уведомляет об этом специальные службы восстановления, выполняющуюся в непривилегированном режиме, о произошедшем крахе и запрашивает у задачи, играющей роль менеджера физической памяти, специальный доступ к кадру памяти отказавшего ядра. Наконец, служба восстановления обеспечивает восстановление ядра в соответствии с определяемой пользователем политикой. Возможные варианты восстановления могут быть следующими:

- игнорирование краха ядра;
- захват памяти, принадлежавшей отказавшему ядру, и возврат этой памяти в обращение путем добавления к кадру памяти работающего ядра;
- обеспечение интерактивного анализа дампа памяти отказавшего ядра и его отладки;
- сохранение дампа в файл для будущего анализа;
- миграция задач из отказавшего ядра в любое работающее ядро;
- фиксирование ошибки на лету для изменения поведения всей системы;
- перезапуск разрушенного ядра с потерей состояния;
- перезапуск разрушенного ядра с сохранением состояния.

Возможные сценарии обработки и восстановления после ошибки могут быть как очень простыми, так и очень сложными, в зависимости от требований, предъявляемых к системе. Самые интеллектуальные сценарии могут обеспечить полное восстановление без какой бы то ни было потери информации и с потерей всего нескольких миллисекунд процессорного времени. Высокий уровень модульности, обеспечиваемый архитектурой микроядра, обеспечивает высокий уровень отказоустойчивости на уровне драйверов и системных служб. Архитектура гетерогенной многоядерной операционной системы позволяет интенсивно использовать дублирование функциональности на разных уровнях. В результате объединения возможностей отказоустойчивости, предоставляемых архитектурой микроядра, с возможностями отказоустойчивости, обеспеченными архитектурой гетерогенной многоядерной операционной системы, и возможностей, связанных с дублированием функциональности, обеспечивается потенциал для создания систем с беспрецедентной отказоустойчивостью, ограниченной только возможностями аппаратного обеспечения.

Архитектура гетерогенной многоядерной операционной системы предлагает не только минимизацию ядра и исключение из его состава максимально возможного объема функциональности в соответствии с принципом минимализма Лидтке, но также и разделение традиционно монолитного централизованного ядра на децентрализованное множество микроядер, обладающих значительной степенью независимости, каждое из которых управляет одним из процессоров или процессорных ядер, входящих в состав компьютерной системы. Таким образом, она предлагает переход от архитектуры операционной системы со статическим централизованным ядром к архитектуре операционной системы с динамическим децентрализованным множеством ядер, в которой операционная система рассматривается как децентрализованная распределенная вычислительная сеть. Такой подход к организации операционной системы обеспечивает ряд преимуществ: простоту реализации ядра, более интенсивное использование кэшей процессора, уменьшение объема блокировок и трафика шины памяти, простоту реализации переносимости между различными типами компьютерных систем, высокий уровень конфигурируемости.

Наряду с уже упомянутыми преимуществами предложенная архитектура гетерогенной многоядерной операционной системы обладает рядом уникальных свойств, которые не могут быть продемонстрированы современными операционными системами.

Первым таким преимуществом является естественная портируемость и поддержка гетерогенных компьютерных систем, которые выступают в роли очень перспективного направления дальнейшего развития компьютерных систем. Можно отметить такие проекты, разработанные в этой области, как процессор IBM Cell и проект гибридного ноутбука, включающего и x86 и ARM-процессор одновременно. Также перспективные системы на базе FPGA, имеющие возможность динамически формировать специализированные процессорные ядра, оптимизированные для решения определенного спектра задач, потребуют возможности быстрого динамического введения в (выведения из) состав операционной системы ядра операционной системы и другого программного обеспечения.

Вторым принципиальным преимуществом предложенной архитектуры гетерогенной многоядерной операционной системы является абсолютная динамичность программного обеспечения. Это означает, что в рамках данной архитектуры любой из компонентов программного обеспечения может быть изменен без необходимости перезагрузки или завершения работы компьютерной системы и, соответственно, без необходимости прерывания обслуживания клиентов компьютерной системы. Это позволяет производить полное обновление программного обеспечения, включая обновление операционной системы и ее ядра, без прерывания обслуживания ее клиентов и изменять функционал, свойства и поведение операционной системы на лету.

Третьим ключевым преимуществом предложенной архитектуры гетерогенной многоядерной операционной системы является высокий уровень отказоустойчивости, основанный на комбинации ключевых преимуществ архитектуры микроядра, децентрализованной автономной природе ОС и превосходных возможностей для дублирования критической функциональности. Возможные отказы программного обеспечения на всех уровнях, начиная с уровня прикладного программного обеспечения и заканчивая уровнем ядра операционной системы, могут быть обработаны без потери данных и только с незначительной потерей времени, оцениваемой несколькими микросекундами.

ЛИТЕРАТУРА

1. *Patterson, D*. Computer organization and design: the hardware/software interface / D. Patterson, J. Hennessy. Morgan Kaufmann, 2008. 912 p.
2. *Liedtke, Jochen*. On µ-Kernel Construction. / Jochen Liedtke // 15th ACM symposium on Operating Systems Principles, 1995. P. 237–250.
3. *Tom Van Vleck.* Unix and Multics / Tom Van Vleck [Electronic resource]. 2010. Mode of access: http://www.multicians.org/unix.html
4. *Rao, P*. United States Patent 7774636: Method and system for kernel panic recovery / P. Rao, L. Varghese. [Electronic resource]. 1995. Mode of access: http://www.freepatentsonline.com/7774636.html